Storypoint Problem Exploration - usergrid

September 6, 2024

1 Storypoint Prediction: Problem Exploration

1.1 Problem Statement

In modern agile development settings, software is developed through repeated cycles (iterative) and in smaller parts at a time (incremental), allowing for adaptation to changing requirements at any point during a project's life. A project has a number of iterations (e.g. sprints in Scrum). Each iteration requires the completion of a number of user stories, which are a common way for agile teams to express user requirements.

There is thus a need to focus on estimating the effort of completing a single user story at a time rather than the entire project. In fact, it has now become a common practice for agile teams to go through each user story and estimate its "size". Story points are commonly used as a unit of measure for specifying the overall size of a user story.

1.2 Problem Formulation

Input: A string of length N that contains a story's name and description $C = \{c_1, c_2, c_3, ..., c_n\}$. For each story, a set of text embeddings that contains features $E = \{e_1, e_2, e_3, ..., e_m\}$ extracted from C has been provided.

Output: A natural number P associated with the story point of that user story

1.3 Dataset Information

Text Embeddings: Text embeddings are a way to convert words or phrases from text into a list of numbers, where each number captures a part of the text's meaning. The dataset has been preprocessed and converted into two kinds of text embeddings. You can choose to work with either of them or both: - Doc2Vec: Input strings are transformed into fixed-length vectors of size 128. These vectors capture the semantic meaning of words and their relationships within a document. - Look-upTable: Input strings are transformed into fixed-length vectors of size 2264. These vectors are obtained via transforming each word in the input strings into an identifier number, then padded to the length of the longest sample.

Dataset Structure & Format: Storypoint Estimation Dataset is stored in 3 folders labeled *raw data*, *look-up*, and *doc2vec*. Within each folder are 3 CSV files for training, testing, validation. Each csv file has the following columns: - `issuekey` : The unique identifier for a story. - `storypoint`: The correct number of storypoint. - An embedding column (`embedding` or `doc2vec`) contains text embedding vectors. The raw data csv will not have this and instead contain two columns with story `name` and `description`.

1.4 Exploration

1.4.1 Raw data exploration

[ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.feature_extraction.text import CountVectorizer
```

Output exploration

[ ]:
```
# Import raw data from the CSV file

project_name = 'usergrid'

all_data = pd.concat([pd.read_csv('data/' + project_name + '/' + project_name +␣
 ↪'_train.csv'),
                      pd.read_csv('data/' + project_name + '/' + project_name +␣
 ↪'_valid.csv'),
                      pd.read_csv('data/' + project_name + '/' + project_name +␣
 ↪'_test.csv')])

print('Check the shape of the dataset', all_data.shape)
```

```
Check the shape of the dataset (333, 4)
```

[ ]:
```
all_data.drop(['issuekey'], axis=1, inplace=True)
all_data.head()
```

[ ]:
```
                                               title  \
0  asset data correctly obey contextual ownership…
1                        expose refresh token rest tier
2               bad geo query returns entire collection
3  empty string doesnt remove entity property nul…
4  fresh admin user token wont work managementuse…

                                         description  storypoint
0  asset data endpoint assetsuuiddata correctly o…           3
1  need add refresh token capability rest tier ma…           3
2  badly formed geo query sent get api returns en…           3
3  users able remove entity properties using eith…           3
4  logging curl post granttypepasswordusernamefds…           3
```

First, let take a look at the distribution of the story point:

Interpretation of Skewness Values:

- **Skewness > 0**: Right-skewed distribution.
- **Skewness < 0**: Left-skewed distribution.

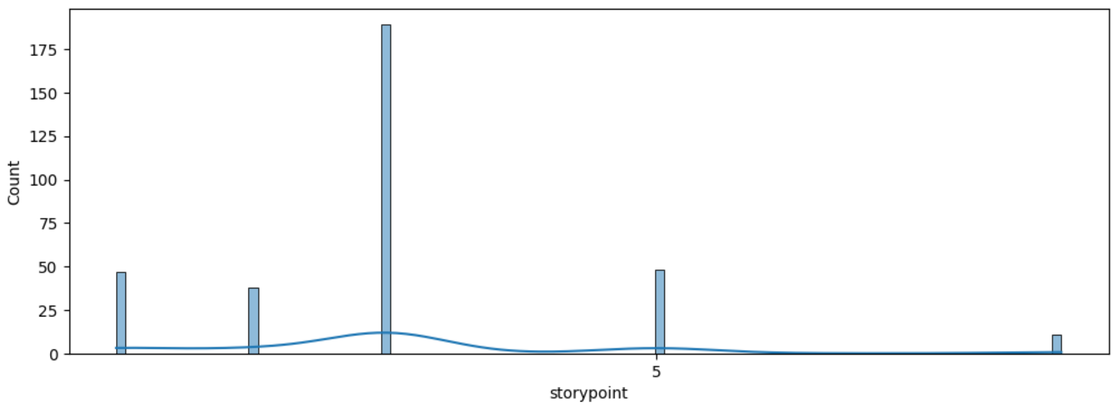
- **Skewness = 0**: Symmetrical distribution (like a normal distribution).

Interpretaion of kurtosis: - **Leptokurtic (Kurtosis > 3)**: The distribution has heavier tails and a sharper peak than the normal distribution. Data points are more likely to produce extreme values. The distribution has a higher peak and fatter tails. - **Platykurtic (Kurtosis < 3)**: The distribution has lighter tails and a flatter peak than the normal distribution. Data are fewer extreme values compared to a normal distribution. - **Mesokurtic (Kurtosis 3)**: The distribution has a similar kurtosis to the normal distribution, indicating a moderate level of outliers.

[ ]:
```
# Draw a histogram of the story points
plt.figure(figsize=(12, 4))
plt.xticks(np.arange(0, max(all_data['storypoint']) + 1, 5))
sns.histplot(all_data['storypoint'], bins=100, kde=True)

print('Skewness:', all_data['storypoint'].skew())
print('Kurtosis:', all_data['storypoint'].kurt())
```

```
Skewness: 1.2353032737625909
Kurtosis: 2.7202275525441997
```



[ ]:
```
tmp = pd.concat([all_data['storypoint'].value_counts(),
                 all_data['storypoint'].value_counts() / all_data.shape[0] *␣
 ↪100],
                axis=1, keys=['Counts', 'Percentage (%)'])
tmp.head(20)
```

[ ]:

storypoint	Counts	Percentage (%)
3	189	56.756757
5	48	14.414414
1	47	14.114114
2	38	11.411411
8	11	3.303303

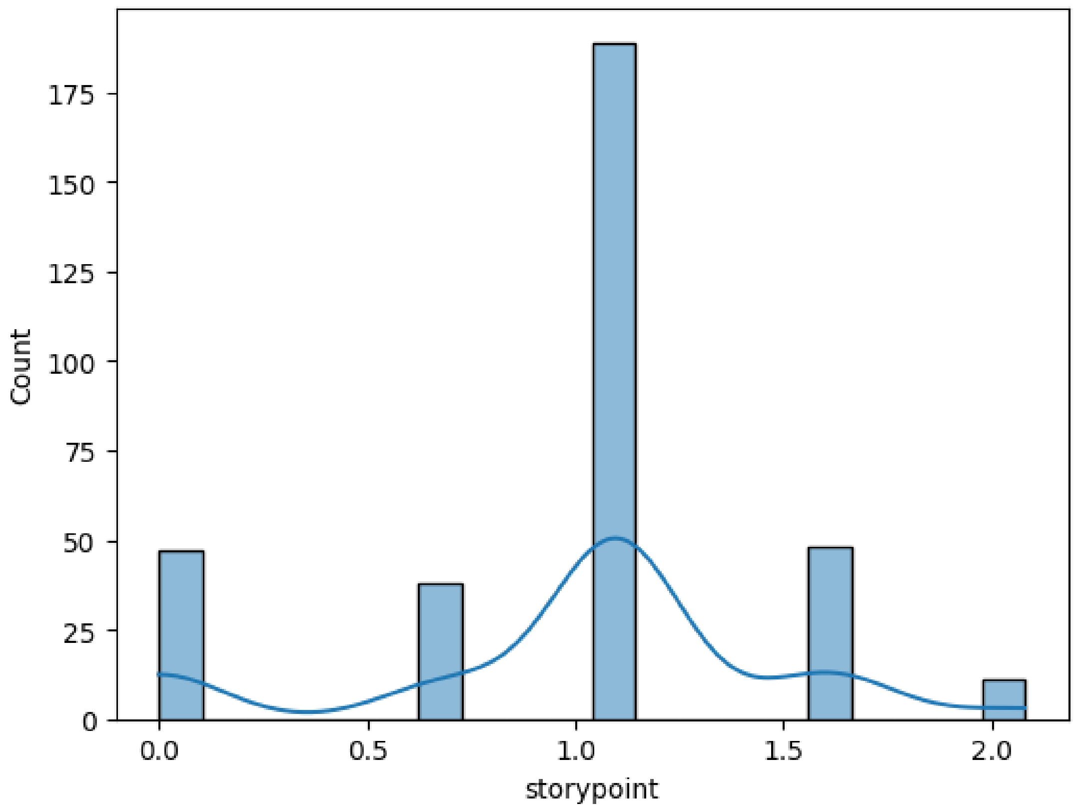
This data is a little bit right-skewed and seperate in 5 groups. Maybe this problem is suitable for a classification method than a regression method

I will try 2 solutions: - Use log-scale on the label - Remove all the examples with label greater than a threshold (20, 30 or 40)

The first solution: logarithm magic

[ ]:
```
sns.histplot(np.log(all_data['storypoint']), bins=20, kde=True)
```

[ ]:
```
<Axes: xlabel='storypoint', ylabel='Count'>
```



[ ]:
```
print('Skewness:', np.log(all_data['storypoint']).skew())
print('Kurtosis:', np.log(all_data['storypoint']).kurt())
```

```
Skewness: -0.5822367330466891
Kurtosis: 0.3649582125623412
```

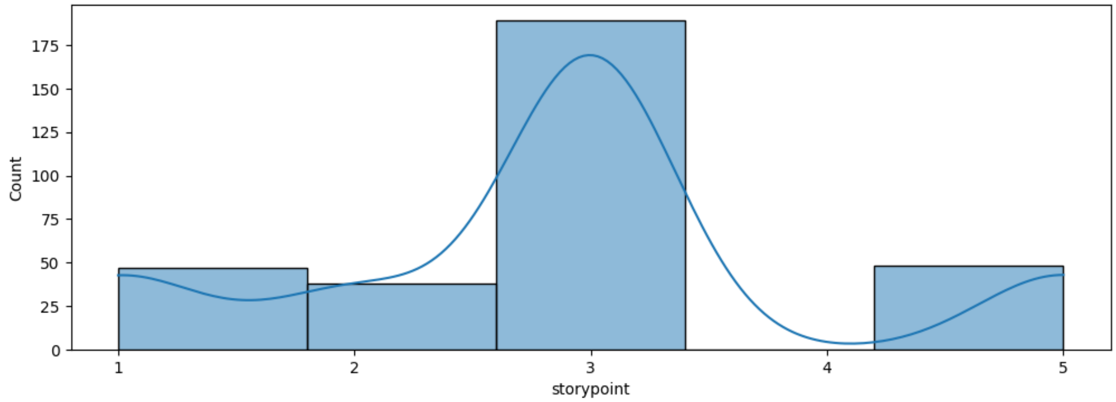
Kurtosis now becomes platokurtosis but skewness is better than before

The second solution: Dismantle and Cleave

[ ]:
```
threshold = 5 # This threshold means that we will take all the examples with␣
  ↪story points less than or equal to 5
```

```
new_data = all_data[all_data['storypoint'] <= threshold]
plt.figure(figsize=(12, 4))
plt.xticks(np.arange(0, max(new_data['storypoint']) + 1, 1))
sns.histplot(new_data['storypoint'], bins=threshold, kde=True)
print('Fitered percentage: ', round(1 - new_data.shape[0] / all_data.shape[0],␣
 ↪2) * 100, '%')
```

```
Fitered percentage:  3.0 %
```



Input exploration The input of this problem is 2 texts: title and description. First we will find some statistics:

[ ]:
```
title_lengths = all_data['title'].apply(lambda x: len(x.split(' ')))
print('Title analysis:')
print('   - Mean length:', round(title_lengths.mean()))
print('   - Min length:', title_lengths.min())
print('   - Max length:', title_lengths.max())

description_lengths = all_data['description'].apply(lambda x: len(x.split(' '))␣
 ↪if type(x) != float else 0)
print('Description analysis:')
print('   - Mean length:', round(description_lengths.mean()))
print('   - Min length:', description_lengths.min())
print('   - Max length:', description_lengths.max())
```

```
Title analysis:
   - Mean length: 6
   - Min length: 1
   - Max length: 13
Description analysis:
   - Mean length: 36
   - Min length: 0
```

```
- Max length: 436
```

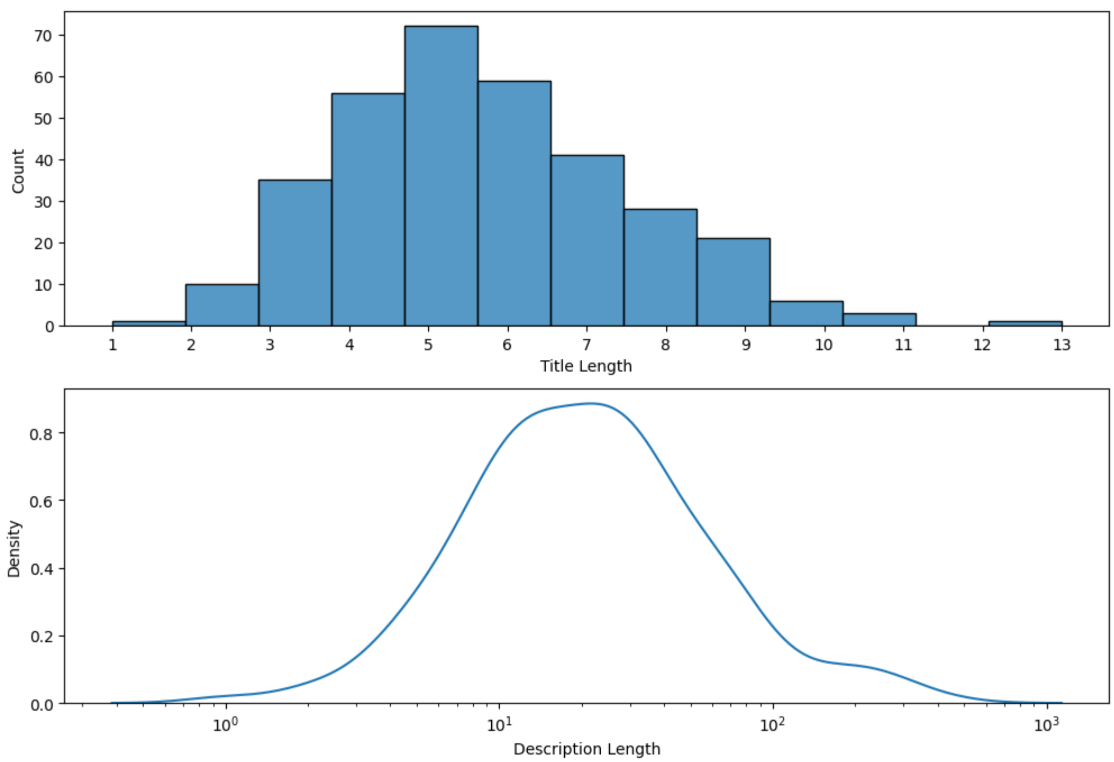
Plot the histogram of the title length and KDE of the description length (exclude 0):

[ ]:
```
plt.figure(figsize=(12, 8))

plt.subplot(2, 1, 1)
plt.xticks(np.arange(0, max(title_lengths) + 1, 1))
plt.xlabel('Title Length')
sns.histplot(title_lengths, bins=max(title_lengths))

plt.subplot(2, 1, 2)
plt.xlabel('Description Length')
plt.xscale('log')
sns.kdeplot(description_lengths[description_lengths > 0])
```

[ ]:
```
<Axes: xlabel='Description Length', ylabel='Density'>
```



I think we should check the correlation between title length and description length:

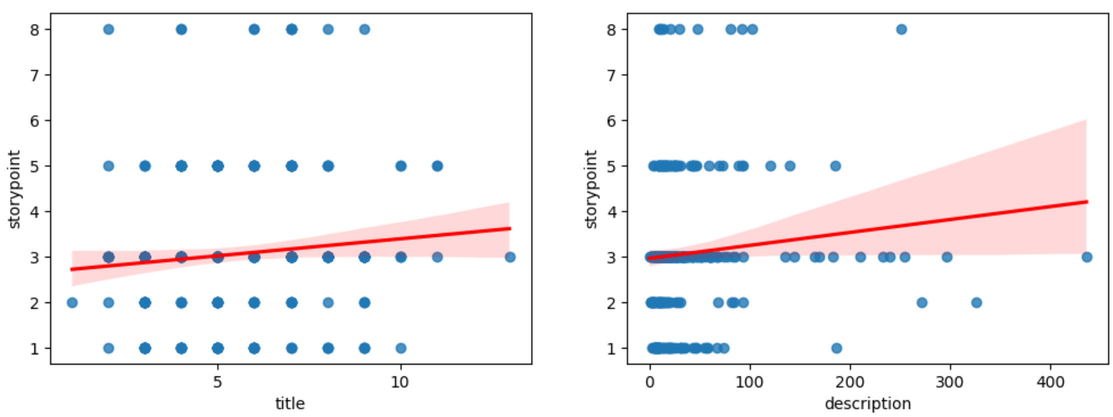
[ ]:
```
plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
```

```
plt.xticks(np.arange(0, max(all_data['title'].apply(lambda x: len(x.split('␣
↪')))) + 1, 5))
sns.regplot(x=all_data['title'].apply(lambda x : len(x.split(' '))),
            y=all_data['storypoint'],
            line_kws={'color': 'red'})

plt.subplot(1, 2, 2)
sns.regplot(x=all_data['description'].apply(lambda x : len(x.split(' ')) if␣
↪type(x) != float else 0),
            y=all_data['storypoint'],
            line_kws={'color': 'red'})
```

[ ]:
```
<Axes: xlabel='description', ylabel='storypoint'>
```



The corelation is having too much deviation on the right plot and a little bit on the left.

Let dive deeper in the input:

Title analysis:

[ ]:
```
count_vectorizer = CountVectorizer()
count_vectorizer.fit(all_data['title'])

dictionary = pd.DataFrame(list(count_vectorizer.vocabulary_.items()),␣
↪columns=['word', 'frequency'])
dictionary.sort_values(by='frequency', ascending=False, inplace=True)
print(dictionary.shape)
dictionary.head(10)
```

```
(828, 2)
```

[ ]:

	word	frequency
138	wrong	827
232	writes	826

```
381  writereview        825
403        write        824
227        works        823
45       working        822
170     workflow        821
25          work        820
30          wont        819
665      without        818
```

Description analysis:

```
[ ]: count_vectorizer = CountVectorizer()
     count_vectorizer.fit(all_data[all_data['description'].isnull() ==␣
      ↪False]['description'])

     dictionary = pd.DataFrame(list(count_vectorizer.vocabulary_.items()),␣
      ↪columns=['word', 'frequency'])
     dictionary.sort_values(by='frequency', ascending=False, inplace=True)
     print(dictionary.shape)
     dictionary.head(20)
```

```
(3003, 2)
```

```
[ ]:                                                   word  frequency
     749                                          zookeeper       3002
     2807                                               zip       3001
     119   ywmtxkhayeeooblzqpxkkaaaaudriyfhegizloihjztoxj…      3000
     1151  ywmtslyneesmfpujajlwaaauvjenpfneqwydbylnuxagsh…      2999
     604   ywmtmzwuiqueescltpguhfegaaaufjuwrktiypwsnvohbv…      2998
     608   ywmtjjacqleesrfvxwfraaaaufffucqvwoyiavxxioeaufnoa       2997
     113   ywmtifzejreeohfgowhrloaaaauizmdwtwqtdewhcikgdy…      2996
     2560                                            yields       2995
     498                                                yet       2994
     2745                                               yes       2993
     2282                                   yattributeaction       2992
     1717                                               xyz       2991
     539                                                xxx       2990
     517                                                xsd       2989
     2281                                    xobjectconcept       2988
     1259                                             xfddd       2987
     1286                                             xfcbc       2986
     1263                                            xeeded       2985
     1287                                            xedccf       2984
     1280                                              xeda       2983
```

Yet I don't find any thing special about the words in input except so many things are bad.

1.4.2 Solving strategies

My first intuitation in this problem is that the hard part is not on the algorithm we use, it is on the **embedding** part. Therefore, in case the given embedded datasets work not properly, I will use a better embedding method which is **Bidirectional Encoder Representations from Transformers (BERT)**. Also, I will try an old way to embedding the text too: **Bag of words**.

In conclusion, I will have 4 ways to embed the text: - doc2vec (already available) - Look up (already available) - Bag Of Words - BERT

About algorithm, I will try all the regression algorithm that may give a good result:

- Ridge Regressor
- Support Vector Regressor
- Random Forest Regressor
- Gradient Boosting
- XGBoost
- Lightgbm
- Blended

Maybe, we can change the problem to the classification problem with 100 labels (desparation confirmed). In the classification problem, I will use: - Support Vector Classifier - Softmax Regression (Multinomial Logistic Regression) - Random Forest - Adaboost - XGBoost

Thanks to the libaries, the implementation of all the algorithm shrinks to its minimum form.

At last, there is still a situation that all of mentioned model don't give a good result. This gamble is thrilling (hopeless).

"But would you lose?"

Nah, I'd win.